AR

PATENT ABSTRACTS OF JAPAN

(11)Publication number : **06-250983**
(43)Date of publication of application : **09.09.1994**

(51)Int.CI. G06F 15/16

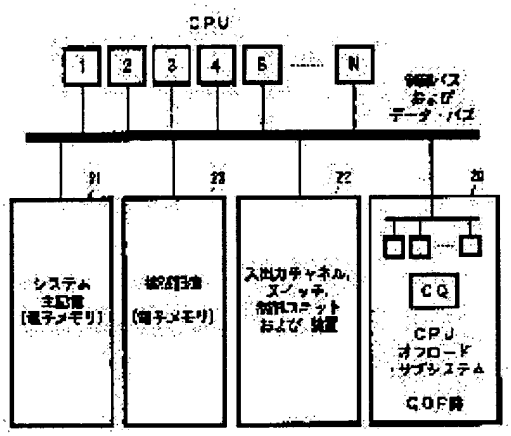
(21)Application number : **06-002345**
(22)Date of filing : **14.01.1994**

(71)Applicant : **INTERNATL BUSINESS MACH CORP <IBM>**
(72)Inventor : **BRENT GLEN A**
DEWKETT THOMAS J
PANNER CHRISTINE R
SCALZI CASPER A

(30)Priority
Priority number : **93 12187** Priority date : **02.02.1993** Priority country : **US**

(54) COMPUTER SYSTEM FOR LOAD BALANCING AND ASYNCHRONOUS DATA MOVEMENT AND METHOD THEREFOR

(57)Abstract:
PURPOSE: To provide load balancing and reconstitution in an asynchronous data movement(ADM) sub-system including plural queue offload processor(QOP).
CONSTITUTION: Each QOP in a CPU offload sub-system 20 has a related queue for receiving a queue element(QE) provided by CPU 1-N in a system. Each QOP executes the QE at the head of the queue. The QOP checks the queue, and processes the waiting QE When the waiting QE is present while the QOP is executing the present QE, the QOP moves the QE from the queue to the queue of the QOP which is not being used, and instructs the QOP to start the processing of the queue. When the empty QOP is not found, the QOP moves the waiting QE to a common queue(CQ). The QOP completes its own QE, and then performs access to the next QE on the CQ, and executes it.



LEGAL STATUS

[Date of request for examination]	14.01.1994
[Date of sending the examiner's decision of rejection]	
[Kind of final disposal of application other than the examiner's decision of rejection or application converted registration]	
[Date of final disposal for application]	
[Patent number]	2587195
[Date of registration]	05.12.1996
[Number of appeal against examiner's decision of rejection]	
[Date of requesting appeal against examiner's decision of rejection]	
[Date of extinction of right]	

DETAILED DESCRIPTION

[Detailed Description of the Invention]
[0001]
[Industrial Application] this invention offers the automatic recovery from the queue processor which generated automatic load balancing between two or more queue processors, and the obstacle, and automatic reconstruction of a subsystem, while the subsystem containing a processor is controlling succeedingly data movement between the electronic storages within an electronic storage without the intervention from an operating system.
[0002]
[Description of the Prior Art] In the conventional computer system, two or more input/output processors (input / output processor) which have the work queue (WQ) of exclusive use were used, and the CPU demand which asks for data movement between an I/O device and computer memory was managed. Each input / output processor WQ operates using a corresponding subchannel identification number using the subset of an I/O device. Each subchannel identification number embraces the path relevant to the equipment with which it is expressed by the queue element in the protected system storage (QE), and the subchannel expresses internally, and is one or more input / output processors. It is assigned in order to use WQ. An I/O device is accessed through various physical paths, and can access some I/O devices through two or more paths. Each path to a certain I/O device can pass along one input/output channel and one control unit (CU), respectively, and can include an I/O path switch. WQ relevant to each input / output processor and it can be served only for QE which has an access privilege to the subset of a path, consequently has a specific path.
[0003] input / output processor manages the workload of the I/O which CPU in a system requires. A subchannel start (SSCH) instruction is executed by every CPU, and it is a certain input / output processor. A work request can be put on WQ. Each CPU demand on WQ contains one queue element (QE), respectively. In QE, it is control block which expresses the I/O device which should be started and specifies data move information. Input/output request cannot be performed unless the physical path to the demanded I/O device containing the channel processor from which it differs for controlling a different physical path becomes usable. Therefore, when either of the components included in the physical path is using it, a demand will be in a standby state. When the control unit (CU) of the physical path to a certain I/O device cannot be used, it is moved to a control unit queue (CUQ) from the WQ, and a demand (QE) stops on this CUQ until CU becomes usable. If CU becomes usable, a demand will be returned to WQ of the origin assigned by the channel processor for the paths, and this channel processor will control input/output operation.
[0004] When a certain input/output operation is completed, assigned input / output processor generates I/O hold interruption by moving to the interruption queue (IQ) to which the QE was assigned by the subchannel, and sending an interrupt signal to all CPUs in a system. The first CPU which can be used for interruption can supervise IQ, and can process one or more hold interruption. Each QE is assigned in order to use IQ shown in the subchannel among eight IQ(s) relevant to a certain interruption subclass. Two or more paths for accessing a certain equipment are defined within the subchannel, the path on a certain WQ becomes under use or use impotentia, and on another WQ, an input / output processor subsystem redistributes work among [of the WQ] some, only when a path is usable. The conventional input / output processor subsystem does not carry out moving QE between WQ(s) which are different in order to use input / output processor of an idle condition. In the conventional subsystem, when QE is the demand to WQ, in order to process this QE, the need had WQ and input / output processor which were assigned to the subchannel. When assigned input / output processor only for WQ(s) had generated under use or the obstacle, unless input/output operation was not started within the channel and the alternate route also existed on another input / output processor, in order to continue execution without discontinuation of QE, QE was unmovable to another WQ. Reconfigurating of input / output processor as a group was only completed by removing the whole group from computer system (when reconfigurating the related portion of computer system). Operation is closed when an alternate route cannot be used. Therefore, input / output processor of the conventional technology WQ and its input / output processor were not used for a workload balance, recovery, or reconstruction.
[0005] In the conventional computer system, when only one path is offered for a certain subchannel, QE relevant to the subchannel ID must stand by in a control unit queue (CUQ), while it stands by by WQ assigned to it while the path assigned to it was using it and the input/output control unit is using it. Since a different subchannel uses different CUQ, CUQ cannot be committed as the central point for equilibrating input / output processor queue work. Therefore, with the conventional technology, QE work request has used only WQ eternally assigned to the subchannel. When the subchannel start instruction (work request) of a large number from one or more CPUs is turned to the subchannel assigned to specific WQ, one or more of

other WQ(s) are empty, and even if the processor relevant to it is an idle condition, all of these work requests must use the WQ. Therefore, in the conventional input / output processor subsystem, a workload may become unbalance very much between WQ(s). moreover, with the conventional technology, when an obstacle occurs in one of input / output processors, a decay-state exists, the input/output request to WQ of the input / output processor is put on a cancel queue, and an alternate route cannot be used -- it is closed when input/output operation is started by the case or its channel In order to repeat the closed input/output request on WQ which was newly able to be assigned and which can be operated, to use a new path, for CPU to bypass input / output processor relevant to WQ and it which generated the obstacle, to re-assign the subchannel to WQ in which other operation is possible, to rerun CPU software from the point of the suitable front in the interrupted application program and to enable it to repeat a CPU demand, intervention of human being may be needed.

[0006]

[Problem(s) to be Solved by the Invention] The key objective of this invention is the interior of the subsystem which performs process off-road from the main processor of computer system, and is offering a workload balance, recovery, and reconstruction. This one example is ADM (asynchronous-data movement). Two or more processors are required of this invention in the off-road subsystem of computer system.

[0007]

[Means for Solving the Problem] this invention manages the processing used by two or more processors, in order to perform efficiently and continuously the work subunit (for example, transfer of a data page) between the primitive position in electronic memory, and the destination position in the same or another electronic memory using the processing and the means of ADM which were indicated by the U.S. patent application 07th/No. 816917 specification (PO 9-90-030) incorporated by reference. In the U.S. patent application 07th/No. 816917 specification, it is ADM. The ADM auxiliary processor containing the single processor for performing a CCW (channel command word) program is indicated. S/390 of subchannel start (SSCH) instructions are used for the CPU interface to ADM. ADM operation is required using a special ADM subchannel. An ADM subsystem performs a CPU demand, and while moving arbitrary numbers of pages which CPU required to the destination position in the same or another electronic memory from the primitive position in electronic memory, CPU can do other work. This subsystem has two or more ADM processors, related queues, common queues, and control-logic mechanisms. Two or more ADM processors can perform simultaneously two or more CPU demands demanded through two or more ADM subchannels. QE relevant to which ADM subchannel and it can also distribute an ADM demand to a subsystem. The ADM subchannel demand which received every ADM processor from arbitrary CPUs of computer system (CEC) can be performed. It is SCH which discriminates the queue element (QE) relevant to a subchannel and it in this although the implied operand of a SSCH instruction is stored in GR1 (general-purpose register 1). It is ID (subchannel identifier) number. A queue element is microcode-ized control block which expresses Subchannel ID with an internal microcode interface. CPU uses this QE as a "mail package", and a CPU demand is sent to the subsystem which performs a CPU demand.

[0008] ADM of one or more [operand / another / of a SSCH instruction] ADM of the beginning of the ADM program which consists of CCW (channel command word) The position of CCW is specified. Each ADM CCW specifies the list of MSB in the memory of a computer (move designated block). Each MSB specifies transmitting-side specification and receiving-side specification. Transmitting-side specification specifies the start address of the primitive position of arbitrary numbers (it specifies in the page count field) of the data in memory of continuation pages (for example, 4KB/(page)). The start address of the destination position in the memory for receiving the continuation page copied from a primitive position is contained in receiving-side specification. CPU demand information is processed, and as for CPU, an ADM subsystem can do other work, while [the electronic memory within electronic memory] controlling the transfer of a huge number of data pages potentially. After an ADM subsystem finishes transmitting all the pages of a CPU demand, it sends I/O type interruption to all CPUs, and tells completion of this CPU instruction. All CPUs receive this interrupt signal and this is accepted under ES / 390 architecture rules existing [of them / one].

[0009] By this invention, an ADM subsystem can process much CPU demands now continuously very much using the automatic recovery from the processor which generated automatic ADM load balancing and the obstacle without interruption over ADM processing, and automatic reconstruction of one or more processors in an ADM subsystem. The ADM load-balancing function, the recovery function, and the reconstruction function are all transparent to CPU processing and operating system software.

[0010] Furthermore, in order that computer system may interface with the I / O subsystem which has input / output processor and WQ by this invention, the same CPU interface is used, and it can interface now with the ADM subsystem which balances a workload as Present CPU is using it. In this invention, it is made the single subsystem which uses the property of this compatibility with an interface, performs operation of both types combining operation of an ADM subsystem, and operation of a I / O subsystem, and can share 1 set of same work queues. Therefore, in this subsystem put together, a queue can hold both an ADM demand and input/output request, and each queue off-road processor (QOP) performs both an ADM function and an I/O function, respectively. However, the common queue in the subsystem put together (CQ) is used only for ADM operation.

[0011] Within the subsystem put together, a queue processor operates as an ordinary input / output processor, while processing input/output request, and while processing the ADM demand, it is operating as an ADM processor.

[0012] However, an ADM function and an I/O function can be offered in the separate subsystem in the same computer system (an ADM subsystem and I / O subsystem). However, generally, in the present system, if an ADM demand and input/output request prepare the subsystem which shares a processor and a queue and which was put together, it turns out that cost-performance is good.

[0013] Moreover, in the system put together, processing priority higher than an ADM demand is given to input/output request on

each queue. because, input/output request is not processed by the processor relevant to a queue (input/output request is processed by the channel processor which controls, different processor, i.e., I / O data move operation) -- it is -- it is because only the very short processing time is needed On the other hand, an ADM demand is completely controlled by the processor relevant to a queue, and other processors are not used. Since an ADM demand uses a processor too much in this way, an average ADM demand needs more [far] processing times than any input/output request, therefore even when I/O traffic is high, the direction of ADM work may consume much processor time. However, input/output request can be processed in the intervals of an ADM work subunit.

[0014] Therefore, although the desirable example of this specification explains ADM operation as what is processed by QOP (queue off-road processor), QOP does not need to be only for ADM operation and should understand that QOP may perform the usual input/output operation on the same queue.

[0015] this invention offers the new method for managing the CPU demand which is managed in a different form from the input/output request processed by the same hardware processor and which asks for ADM service. The same hardware processor performs different processing about an ADM demand and input/output request.

[0016] Therefore, this invention supports the same queue and the data control subsystem which uses the group of a queue processor about both a respectively separate queue, the data control subsystem which uses the group of a queue processor, and an ADM demand and input/output request about an ADM demand and input/output request.

[0017] Unlike an I/O subchannel, an ADM subchannel does not have a path limit at all. All the paths between the electronic storage in electronic storage are accessed, the perfect flexibility for processing the ADM demand from arbitrary CPUs among all ADM queues and ADM processors is given, and every ADM processor can be made accessible from arbitrary demands through arbitrary ADM subchannels at all electronic storage. On the other hand, I/O subchannel path control needs to pass input/output request to the input/output processor in which an input/output processor has one of the specified usable paths. Then, this demand is given to a path processor (channel processor), and input/output operation is completed. If the path control about ADM operation is removed, an ADM processor can complete ADM operation, without needing a path processor.

[0018] this invention offers two steps of workload balances. According to the first workload balance, an ADM subsystem comes to distribute a CPU demand equally between ADM processors. An ADM subsystem controls allocation to the specific ADM processor of a specific ADM demand. The workload balance of a culmination is offered within a subsystem. Since a CPU demand may change from a single data transfer to extremely a lot of transfer, the subsystem execution time may change sharply. As long as it moves work to the ADM processor of an idle condition continuously from an ADM processor in use and a demand exists in a subsystem as workload execution advances within a subsystem, an actual workload change is completely equilibrated by maintaining so that all ADM processors may use it continuously. If the queue and processor which are shared by a separate queue and the processor row are used, when a queue receives an ADM demand and the processor relevant to the queue will not use [be / it] it, the processor performs the ADM demand immediately. When a queue receives an ADM demand and the processor relevant to the queue is using it, in this invention, by making an waiting ADM demand transmit to the queue of other processors which are not used for a processor in use and which they are not, an ADM workload balance is offered and this processor performs an ADM demand immediately. When all the processors relevant to a queue are using it and one of queues receives an ADM demand, it stops between the subunits of work and a related processor moves the ADM demand to a common queue (CQ). CQ, it is processed by either of the processors relevant to a queue when each processor relevant to ** and a queue looks for the new work request which should complete and perform a demand. A processor looks at CQ header and investigates whether CQ is empty. the case where CQ is not empty -- the processor -- immediate -- an waiting ADM demand -- CQ -- since -- it moves to the queue of the processor and this ADM demand is performed from this queue

[0019] Therefore, this invention redistributes an waiting ADM demand continuously between the queue and processor, and all ADM demands are made to perform quickly by the processor relevant to a queue. Therefore, an ADM demand can make the execution complete in front all the time rather than the processor relevant to the queue becomes usable because of the processing. An AMD demand is because what is necessary is just to stand by between the minimum in a common queue until the processor which is immediately performed by arbitrary usable processors, or becomes usable at the beginning while all processors are using performs. A perfect workload balance is brought about about all ADM demands by redistribution of this work request. There is almost no interference to the input/output request processed by the same queue and the same processor. It is because the input/output request put on the queue is processed by the processor relevant to a queue at the same time the present subunit of work is completed.

[0020] Furthermore, this invention offers each ADM recovery function about QE, when QE is used as an ADM demand on a queue. An ADM recovery function is realized by the checkpoint processing field in QE, and the recovery for using it. The checkpoint processing field saves the information in QE, in order to trace the page of the last successfully moved during execution of an ADM demand by the processor which generated the obstacle. When an obstacle occurs during execution of page move operation at one of processors, this processor sends the signal which tells all other processors relevant to a queue about the fault condition and identifier. The processor relevant to the queue which detected this trouble back signal first turns into a recovery processor by accepting a trouble back signal. A recovery processor resets the processor which generated the obstacle, adds 1 to the retry count of the queue header relevant to it, and establishes the conditions which make operation retry from the work subunit of the last which completed the count successfully to the processor which generated the obstacle compared with the maximum permissible retry count when smaller than it. When that is not right, the processor which generated the obstacle is stopped, it is removed from the subsystem which can operate, and the work is redistributed to other processors through subsystem workload balance processing.

[0021] In case recovery is tried, a recovery processor re-establishes execution of the demand from the work subunit of the last completed successfully until it accesses the checkpoint processing field of QE which the processor which generated the obstacle is processing, it uses the data in the checkpoint processing field and an imperfect ADM demand is completed successfully. When the processor which generated the obstacle cannot be recovered, the QE element is put on the related queue of a recovery processor. or this recovery processor completes this work request after that -- or this QE -- CQ, it can be alike, it can place and the processor which completed the on-going demand first can be made to complete it Operation begins from the state where checkpoint processing was carried out, in any case. This is which ADM under execution on a related queue. Since it can perform also by QE until it completes successfully by the arbitrary processors in a subsystem, it differs from radial transfer. It is searched for QE in order to determine the queue allocation to the new demand to the queue which remains after recovery (based on CPU), and to change this in being required, when a certain processor is removed by recovery.
[0022] If this recovery function is used, it is a form transparent into the remaining portion of the computer system of the exterior of this subsystem, and ADM operation can be recovered completely, without interrupting the work under execution by the subsystem of a parenthesis. Since this subsystem is operating using a fewer processor while a demand load is high, the speed of subsystem operation may fall slightly.
[0023] this invention also offers the ADM dynamic-reconfiguration function which can be used at any time, in order to remove arbitrary processors or to add a new processor to a subsystem, as long as at least one processor which can be operated remains in the subsystem. The rediscount reliance of QE to the queue which remains after reconstruction, and its processor is contained in reconstruction. In this invention, each header of the queue (for [which is removed] processors) removed in reconstruction operation is marked by setting to a impossible state of operation the field which can be operated. The demand under hold on the queue made into impotentia of operation (QE) is altogether moved to other queues (a common queue is included) which can operate freely. The processor assigned as a reconstruction processor which processes the command in a reconstruction command can perform this QE move operation. Moreover, in this invention, the header of each queue (for [which is added] processors) added in reconstruction operation is marked by setting to a impossible state of operation the field which can be operated.
[0024] The processor relevant to a queue searches for all QE(s), and determines the queue allocation to the queue which remains after reconstruction, and reconstruction processing needs to change this, when required. When removing a processor from a subsystem logically (a processor is deleted), the queue is also removed and a reconstruction processor re-assigns QE of the deleted processor to the processor queue which remains. In the addition of a processor queue, a subsystem can re-assign a part of QE to each new processor queue from the front existing queue. QE under hold to the added processor by the workload balance function which carries out the increment of related Q(queue) ID, and the function which assigns work to an unused processor is assigned.
[0025] Thus, in case workload balance operation of this invention, recovery action, and reconstruction operation are performed, an ADM demand is redistributed to the queue which can be operated, and a common queue, without needing the intervention of other portions of computer system, or the software in computer system (the intervention by the software operating system also including that it is not required).
[0026]
[Example]
CPU operation (drawing 1)
Drawing 1 is drawing containing the CPU off-road subsystem 20 which carries out off-road one of the asynchronous-data move management activities to the CPU off-road subsystem 20 from two or more CPU1 or N, and moves the page of data among these in the electronic memory 21 or the electronic memory 23, and I / O subsystem 22 containing an input/output channel path, an I/O switch unit, an input / output control unit, and equipment showing computer system. The electronic memory 21 is the system main storage 21 (MS may be called), and the electronic memory 23 is extended storage (ES may be called).
[0027] S/390 of subchannel start (SSCH) instructions are used for each CPU, and it puts the queue element (QE) assigned to the subchannel on the queue of a subsystem. To input/output request, it operates by the method indicated by 7201 and a SSCH instruction operates to an ADM demand by the method by which was indicated by the 07th/816917 (PO 9-90-030) number specification of the EnterpriseSystem Architecture/IBM publication and "390 Principles of Operation" data number SA22-U.S. patent application which indicated above, and the patent claim was carried out.
[0028] By both [of input/output request and an ADM demand] cases, ORB (RB of operation) specified by the second operand is accessed by CPU execution of a SSCH instruction. A subchannel identification number is specified by the general-purpose register 1 (GR1). A SSCH instruction uses the priority indicated on these specifications which access the queue in the CPU off-road subsystem 20 which called the microcode in the CPU and was assigned to each subchannel discriminated, and are in each queue, and carries out the chain of the QE relevant to a subchannel to the queue assigned to QE. Many fields in the header of a queue are set to the address of the present time edge (last) QE of the queue, and the address of the present upper limit (the 1st) QE of the queue.
[0029] Although each QE of arbitrary queues expresses "the unit of work" to the CPU off-road subsystem 20, this is a demand for which it asks so that a shown number of pages may be transmitted between the primitive address discriminated within the QE, and a destination address to the queue off-road processor (QOP) relevant to the queue containing the QE.
[0030] Execution of a SSCH instruction is completed after sending the signal which tells that the chain of the accessed QE was carried out as the last QE by the discriminated queue, and the chain of the new queue element (QE) was carried out to the queue at QOP to which CPU relates. When QOP is not using [be / it] it then, the QOP accesses immediately QE of the beginning of the queue (in the case of a desirable example) relevant to it, and starts the execution. When a new element is the only element of

the queue, QOP which received the signal accesses new QE and acquires the following work unit. When QOP is using it, the QOP discovers a new demand by testing, after completing each work subunit.
[0031] Operation of a subsystem (drawing 1 and drawing 2)
The CPU off-road subsystem 20 of drawing 1 includes 1 set of queue off-road processors (QOP), the queue (a "QOP queue" is called on these specifications) relevant to QOP, and a common queue (CQ). CPU puts a demand on a QOP queue, and QOP takes out these demands from each QOP queue, and performs these, or moves to other queues. Each CPU demand is put on a queue in the form of the queue element (the address is carried out within a queue) (QE is called) by which the chain was carried out to the queue. The chain of the ADM demand or input/output request from CPU can be carried out to a queue by making a subchannel into a queue element (QE). Therefore, QOP is an off-road subsystem processor which answers the CPU demand put on a queue and performs asynchronous-data move processing. A CPU demand is an ADM demand which asks for moving N pages among these in input/output request, the electronic memory 21, or the electronic memory 23. namely, every -- QOP -- radial transfer -- be -- ADM processing -- be -- one or more processings which each received CPU demand needs, respectively are performed ADM execution control, load balancing, recovery, and reconstruction are included in ADM processing which QOP performs.
[0032] The CPU off-road subsystem 20 is shown in drawing 2 in detail. Each queue element has the field set by initial setting so that it may be shown which demanded subchannel operation shall express between input/output request and an ADM demand. CPU puts input/output request or an ADM demand on the soffit of each QOP queue. Each QOP finds the following work unit by accessing QE in which tab control specification is carried out by the address in the "Top Q PTR (queue head pointer)" field of the header of the queue relevant to it. Work units are the I/O QE or ADM by which a chain is carried out within a queue. It is expressed by either of the QE(s). The queue will be empty if the address of the header is included in the "Top Q PTR" field. When it is not empty, a processor accesses QE in the head of the queue, and changes the content of the "Top Q PTR" field into the address of the following QE within the queue. When there is no following element, a processor writes in the header address. The present work unit has a chain canceled of every queue, when it is going to perform it.
[0033] When input/output request or an ADM work unit is completed, QOP looks at a common queue (CQ) header, and investigates whether CQ is empty (when in other words one whole ADM demand is completed). the case where CQ is not empty -- QOP -- CQ, the inner head QE is moved to the head of the queue of QOP, and it returns to queue processing the case where QE is moved -- or -- CQ, when it is alike and there is no QE, QOP looks at the queue header and investigates whether it is empty When a queue is not empty, QOP performs QE in the head of the queue. All new CPU demands are put on the soffit of a queue. Thus, by putting QE on each queue and taking out from there, the priority of execution of QE in all queues is maintained, and it tries to process QE in order near in order of arrival as much as possible.
[0034] The radial transfer performed by QOP is ES of IBM/9000. M900 system, S/3090 former system, etc. are the ordinary input / output processor processings performed by ordinary input / output processor (input/output processor) looked at by the mainframe of IBM.
[0035] The ADM processing performed by QOP is the theme of this invention. The common queue in the CPU off-road subsystem 20 (CQ) is used only for ADM processing, and is usable by all QOP(s).
[0036] QOP inspects [whether there is an ADM demand under hold or input/output request under hold, and], after completing a work unit, and after completing each work subunit during work unit execution. QOP inspects a queue by investigating whether the header of the queue was seen and the new demand was put on the queue. When a queue has a new demand, QOP inspects whether it is an ADM type thing, or it will not be an I/O type. When it is an I/O type thing, by transmitting the CPU demand to a channel processor, QOP processes this and returns to the ADM work on a subsystem queue.
[0037] When a new demand is an ADM type thing, QOP will try to pass the new demand to an unused processor if an unused processor is found. this -- CQ, a pointer is inspected during the QOP use in a header, and it is carried out by judging whether there is unused [QOP] Then, QOP which conducted this inspection transmits a new ADM demand (QE) to the soffit of a queue unused [QOP] from the head of the queue, and sends a signal to unused [QOP]. the case where QOP which inspects cannot find an unused queue -- this QOP -- the new ADM demand -- CQ, it is alike and returns to its processing Then, whatever what was being processed when it stopped, QOP which inspected continues the processing. QOP inspects CQ header for whether it is an empty state, when it will be in an idle condition. When CQ is not empty, QOP takes out top CQ demand and moves to the head of the queue. This CQ demand is taken out from the head of the queue as this next work unit of QOP. When CQ is empty, QOP inspects [whether there is any work which should be done on a degree, and] the queue of itself. When there is anything no], QOP shifts to a standby state.
[0038] The usual one of QOP of operation and QOP remove the top subchannel QE from the queue relevant to it, perform the QE, and work one unit. The subunit of a specified number of data is transmitted between two positions specified within the queue element by execution of QE. These two positions are the address in the same electronic storage, or the address in two different electronic storages in a system. These operation follows instruction of the 07th/816917 (PO 9-90-018) number specification of the U.S. patent application. For example, "work units" is a 10000 pages data transfer between MS and ES, and a 38-page data transfer between two positions in MS. That is, a transfer of each page is the example of a "work subunit." A work unit becomes equal to a work subunit, only when the work unit specifies a single subunit transfer.
[0039] QOP is stopped whenever it completes a work subunit. It investigates whether QOP looks at the queue head pointer of the queue header, and QE which is standing by by the queue is between this pause. Waiting ADM When there is QE, stopped QOP changes from subunit transfer operation temporarily, and looks for QOP under un-using it within a subsystem. When QOP under un-using it is found, stopped QOP moves waiting QE to the QOP queue under un-using it, and the signal which tells that QE is in the queue relevant to it at QOP under the un-using it is sent. When it proves that all QOP(s) are using it, stopped QOP moves

waiting QE to the soffit of a common queue (CQ).
[0040] When work does not exist in a CQ or relation queue by the usual processing, QOP is I/O QE or ADM. It stops at an idle condition until it receives a notice that QE was put on the queue.
[0041] It is ADM between a QOP queue and a common queue. As long as CPU has put QE on the queue of either of the subsystems by performing load-balancing redistribution of QE, all QOP(s) are kept continuous while in use. According to this method of operation, compared with ADM demand processing of the subsystem which has the queue which is not a load-balancing formula, the maximum processing speed is obtained about all the work supplied to a subsystem.
[0042] Queue element (QE) (drawing 3)
Drawing 3 is drawing showing QE which is control block memorized by the protected electronic memory mechanism relevant to a subchannel.
[0043] The subchannel itself which is control block which has the pointer field for carrying out the chain also of the control block (a subchannel being pointed out) relevant to a specific subchannel to a queue various type is sufficient as QE used by this invention.
[0044] Each QE in this example "lock (lock)", "subchannel ID (subchannel identifier)", "type (type)", "busy ID (under use identifier)", "Current Q ID (the present queue identifier)", "Current Q PTR (the present queue pointer)", "Current Qchain PTR (the present queue chain pointer)", "assigned Q ID (assigned queue identifier)", Each field of "Checkpt Subunit Data (checkpoint subunit data)", "Channel Program Address (channel program address)", and "status (situation)" is included.
[0045] In order to change one or more fields, while accessing the QE, in order to warn other QOP(s) and CPUs of the "lock" field, it is set to ON.
[0046] The identification number of a related subchannel is stored in the "subchannel ID" field.
[0047] For the "type" field, the QE is I/O QE and ADM. It is shown which [of QE] it is. When it is not which, either, it is not the demand processed by the QOP subsystem.
[0048] The "busy ID" field discriminates QOP under access to this QE now. The "current Q ID" field discriminates the present queue containing this QE.
[0049] The "current Q PTR" field carries out the address of the queue header of the queue containing this QE. This field is cleared while this QE does not exist in all the queue, either.
[0050] The "Current Q chain PTR" field carries out the address of the QE performed next within this queue. this QE -- the last within a queue -- when it is QE, this field has a special value This field is cleared while this subchannel does not exist in all the queue, either.
[0051] The "assigned Q ID" field discriminates the QOP queue of the point which CPU assigns this QE.
[0052] In the "Checkpt Subunit Data" field 1) The effective-bits (V) subfield set to ON when all the subfields in "Checkpt Subunit Data" are effective, 2) The present ADM within the present ADM program A CCW subfield including the address to CCW (channel command word), 3) The present ADM An MSB subfield including the address to the present move designated block (MSB) which specifies the subunit move parameter of CCW, And two or more subfields containing the moved subunit subfield which shows the number of present of the subunit successfully moved about 4 present MSB are contained. ADM CCW operation and MSB (move designated block) operation are indicated by the U.S. patent application 07th/No. 816917 specification quoted previously, and the patent claim is carried out. The address of the ADM channel program obtained from ORB of a SSCH instruction call is stored in the "Channel Program Address" field of QE. In the case of input/output request, the address of a required input/output channel program is stored in this field. In an ADM demand, the address of a required ADM program is stored in this field. It is ADM in memory as a "ADM program". It is the list of CCW.
[0053] A work unit is completed, and when put on an interruption queue, the status information about an ADM work unit reported to the requiring agency CPU is stored in the "status" field.
[0054] The header of a QOP queue (drawing 4)
Drawing 4 is drawing showing the important field used by the header of each queue relevant to QOP. Each queue can access the header relevant to it at any time, when necessary [to have a header for carrying out support of the queue to the known address for all CPUs and QOP(s), therefore for those CPUs and QOP(s) to access a queue]. Many fields used in this embodiment, i.e., the lock field, the enable (operation is possible) field, the retry count (retry count) field, the top Q PTR (queue upper-limit pointer) field, and the bottom Q PTR (queue soffit pointer) field are included in each header of a QOP queue. The lock field is surely set, when changing the field in this header. The enable field is set to ON when operation of a related queue is enabled, and when a related queue is made into impotentia of operation, let it be the field of the single bit set to OFF. The present number of errors with which related QOP encountered in the time window defined in advance is stored in the retry count field. The address of QE (QE which has the priority highest by the queue now) in the head of a related QOP queue is stored in the top Q PTR field. This is QE performed next by this queue. At the time of empty (QE is not included), the address of the header is stored in the top Q PTR field for a queue.
[0055] It is in the soffit of a queue and the address of QE which has the priority minimum by the queue is stored in the bottom Q PTR field. This is QE performed at the end among QE(s) which are in the queue now. The bottom Q PTR field is accessed in order to put the next QE on a queue.
[0056] The header of a common queue (drawing 5)
Drawing 5 is drawing showing the field in the header of the common queue (CQ) which all QOP(s) in a subsystem use. ADM QE -- CQ, it is alike and is placed CQ, the field used in this embodiment, i.e., the lock field, QOP0 busy PTR (under QOP0 use pointer) or the QOPy busy PTR (under QOPy use pointer) field, the top Q PTR field, and the bottom QPTR field are included in

a header (CQH).
[0057] The lock field is surely set, when CQH is changed. ADM performed by each QOP in QOP0 busy PTR or the QOPy busy PTR field now, respectively (removed at the end) The address of QE is stored. QE address of this field shows a state during each use of QOP. Each content of QOP0 busy PTR or the QOPy busy PTR field is cleared when QE is not processed by each QOP (to empty).
[0058] The top Q PTR field carries out the address of the QE (QE performed next by the queue) in the head of CQ, and when CQ is empty, it carries out the address of the CQH.
[0059] The bottom Q PTR field carries out the address of the QE in the soffit of a queue. this -- present -- CQ, it is QE which is alike and is performed at the end among existing QE(s) This is the field accessed in order to place QE of CQ ****.
[0060] The CPU demand which asks for subsystem service (drawing 6)
Drawing 6 is drawing showing the processing which CPU uses, in order to put a work unit on one of QOP queues. In this processing, CPU carries out the chain of the QE to the soffit of the QOP queue determined by queue allocation of the "current Q ID" field of QE. A CPU demand can also require an I/O work unit and ADM work unit.
[0061] At Step 50 of the beginning of drawing 6 , one of CPUs accesses S/390 of "subchannel start (SSCH)" instructions, and the demand about a work unit is given to a QOP subsystem. The first operand of this SSCH instruction is a subchannel identification number specified by the general-purpose register 1 (GR1). The processing shown in drawing 6 is similar to the ordinary processing used for starting of an I/O device by the mainframe of IBM. In the case of the latter, CPU makes the subchannel specified to be the input / output processor queue specified by QE a queue element (QE), and carries out a chain, and the signal it is directed to input / output processor (input/output processor) relevant to the queue that processes a new queue element is sent.
[0062] Next, the usual SSCH instructions including the exception test under execution are executed in the form performed with the conventional technology by Step 51. QE is tested by the microcode about the ability to reach [whether it is effective and] and operate in early stages of executive operation. When it is shown that it cannot operate by these tests or the QE is not effective, it progresses to Step 52 through an "exception" path, and processing is ended. When it is shown by these tests that the QE is effective and it can operate, it progresses to Step 53 and CPU carries out the chain of the QE to the soffit of the queue specified in the "assigned Q ID" field of QE there.
[0063] Next, at Step 54, a CPU microcode supplies the signal which shows what new QE was put on QOP relevant to the assigned queue for by the queue. With this signal, if QOP is not [be / it] under use by execution of other work, it can process this QE immediately. The subchannel operand of this SSCH instruction is expressed by QE on the assigned QOP queue at this time.
[0064] CPU execution of a SSCH instruction sets the condition code of 0 at Step 55 (CC=0), and ends it by showing in a CPU program that this instruction was completed successfully. A CPU program can execute the next instruction freely at this time, and the next instruction does not need to have this SSCH instruction and relation.
[0065] Processing of a queue element (drawing 7)
It progresses to step 78H from a <A HREF="/Tokujitu/tjitemdrw.ipdl?N0000=237&N0500=1 E_N/;?9=:?67YES" path, and a CHECKQUEUE process is ended. When it is not empty, it progresses to step 78C from a "NO" path, and Head QE is removed from the queue (it removes). Next, QE removed by step 78D is ADM. It tests whether it is QE. When that is not right, this is I/O QE, progresses to step 78P from NO path, and performs the usual radial transfer.
[0079] This is ADM. When it is QE, another QOP which can perform QE demand by which standby release was carried out by progressing to step 78E from a "YES" path is looked for. The present QOP should care about that the work unit which has not been completed is performed and it is under use now, when you stop at the time of the end of the last subunit in order to perform a "CHECKQUEUE" process during execution of a work unit. Therefore, the present QOP cannot perform this QE by which standby release was carried out at this time.
[0080] By step 78E, when it turns out that another QOP is an idle condition (since ADM work is not done at all), and operation of a queue of it is enabled, it progresses to step 78F from a "YES" path, and this removed QE is put into a QOP queue unused the / QOP]. Then, what the present QOP was made to send QE signal to unused [QOP], and QE was put on it for by the queue by step 78G is shown in unused [the / QOP]. Then, it returns to the EXECUTE process of drawing 8 , and progresses to Step 73.
[0081] When QOP whose operation was enabled by the idle condition by step 78E is not found at all, it progresses to step 78J, and when the QE has effective Checkpt Subunit Data, the QE is put into the head of CQ, when it does not have effective Checkpt Subunit Data, it puts into the soffit of CQ, and it returns to the process EXECUTE of drawing 8 after that, and progresses to Step 73.
[0082] The GETWORK process from a common queue (drawing 10)
When a "GETWORK execution" step is reached in drawing 7 , it branches and goes into the "GETWORK" process of Step 80 of drawing 10 . In a "GETWORK" process, a common queue header (CQH) is accessed, the "Top Q PTR" field is examined, and it judges whether CQ is empty. In the case of empty (QE is not included), the address of CQ header is included for CQ in the contents of the "Top Q PTR" field. When the effective address is not included there, it returns to Step 67 of drawing 7 at Step 83. However, when the effective address is included there, it is Step 82, and the address is used, CQ shell head QE is removed, the following QE of CQ is acquired, and the QE is put into the head of a QOP queue.
[0083] Then, it progresses to Step 67 of drawing 7 at Step 83.
[0084] Reconstruction processing (drawing 11)
An ADM subsystem can be reconfigurated, when human being's system operator inputs a command into computer system and

removes or adds one or more QOP(s) which have the queue related, respectively. This may be generated when carrying out the subdivision rate of the multiprocessing system to two or more independent images or systems. Each QOP must still [of the system portion after the reconstruction in which the QOP exists physically] be [a part of]. An on-going demand and the demand which already stood by must be processed by the portion which resides permanently after the operating system which started it among systems reconfigurating. An on-going demand must be re-assigned to QOP which remains in the portion on QOP which separates from the system portion into which the operating system under present execution continues execution there. CQ, it is left behind as a part of ** and remaining portion. When reconstruction includes addition of the element before removed by reconstruction, additional QOP becomes a part of QOP subsystem which can operate. The CQ shell QE will be removed and Reconstruction QOP will send a signal to added QOP, if QE exists. CQ, when QE turns up, added QOP waits for a signal.
[0085] Each reconstruction command is supplied to the ADM subsystem in computer system from either CPU or a service processor. In order to call a reconstruction command a QOP off-line change command and a QOP online change command on these specifications, and to remove or to add one or more QOP(s) of a subsystem, an operator can input them. In this example, one QOP is added to a QOP subsystem by the QOP online change command, and one QOP is removed from a QOP subsystem by the QOP off-line change command.
[0086] Drawing 13 is drawing showing QE signal / recovery / reconstruction bus 110 connected to all QOP(s) in a subsystem. QE signal / recovery / reconstruction bus 110 is connected also to the reconstruction signalman stage 111, and an operator can supply a reconstruction command signal to all QOP(s) through this.
[0087] A QOP off-line change command and a QOP online change command may be dynamic commands which can be published and executed while using a subsystem, and a subsystem may be what operating state, when QOP receives a command. That is, the off-line change command which removes a certain QOP can be executed, without making destruction of a QOP subsystem clear to the remainder of computer system, while the QOP is performing QE. Moreover, the command is which ADM. It does not have a bad influence on the result of execution of a QOP work unit, either.
[0088] While the QOP is performing the work unit, in order to delete the QOP from a subsystem, when an off-line change command is published to a certain QOP, it is used in order that the "Checkpt Subunit Data" field of QE may control correctly the change of the execution of QE to QOP to which after execution of a reconstruction command consists of QOP deleted with QOP in which the same operation for operating systems is possible.
[0089] When a certain QOP is added or deleted by the reconstruction command (it is carried out in this example like), an operator can use a series of commands, and can add or delete a desired number of QOP(s). However, in order to process a multiplex QOP command, it is clear that a number of QOP(s) specified by the single reconstruction command can be removed or added by repeating only the number of times which had the reconstruction processing to one QOP specified.
[0090] The command information sent on QE signal / recovery / reconstruction bus 110 of drawing 13 specifies the identifier of performing which of additional operation and deletion operation, and QOP added or deleted. QOP which discovered the reconstruction signal first performs reconstruction processing. Any QOP which remains being able to operate after completion of reconstruction processing is sufficient as this.
[0091] This command information is detected from QE signal / recovery / reconstruction bus 110, and is memorized by the reconstruction QOP discriminated from discriminated QOP (Addition QOP or Deletion QOP is called) which it is going to add or delete.
[0092] Therefore, one of the QOP(s) which discriminate QOP which it is going to remove as "deletion QOP", and specify it, and remain in a subsystem is used for a QOP off-line change command as "reconstruction QOP."
[0093] Every QE within the queue relevant to Deletion QOP is moved to the queue relevant to Reconstruction QOP from the queue relevant to Deletion QOP. QE is not moved when a deletion QOP queue is empty. However, when a deletion QOP queue is not empty, QE of Deletion QOP is moved to a reconstruction QOP queue. QE move processing is performed by Reconstruction QOP in the desirable example.
[0094] In QOP online change processing, QOP is added to a subsystem. It is that QOP added is physically added to a subsystem, before performing processing of drawing 11 . This additional command discriminates and specifies QOP which it is going to add, and Reconstruction QOP performs additional processing. In this processing, the header of the new queue relevant to QOP added is set up. As for this QOP added, the ID is written in a header by Reconstruction QOP. an online change command -- QE -- CQ -- the case where it is come out and suspended -- CQ, QE is moved to the queue of QOP by which the shell addition was carried out If operation of the queue of added QOP is enabled, it is begun automatically to assign work to the queue of added QOP for the "assigned Q ID increment" of Step 67 of drawing 7 .
[0095] Drawing 11 is drawing showing the processing performed by QOP which accepts a reconstruction signal. It goes into Step 90, after receiving the QOP change command for adding or deleting one QOP. Then, it judges which [of an online change command and an off-line change command] it is from this command actuating signal at Step 91.
[0096] In the case of an off-line change command, it progresses to Step 92 in "deletion" path. At Step 92, the enable field of the queue header of Deletion QOP is set to a impossible state of operation. When all the subchannels that are Step 92, next are in the queue of Deletion QOP are moved to the queue of Reconstruction QOP and Reconstruction QOP returns to an idle condition, a signal is sent to Reconstruction QOP. Finally at Step 92, Deletion QOP is stopped. next, the step 93 -- CQ, the "QOP Busy PTR" field of the inner deletion QOP is accessed, and this will be read if there is the address of "the subchannel by which checkpoint processing was carried out" which was being performed when Deletion QOP was stopped at Step 92 in the field
[0097] QE by which checkpoint processing was carried out is removed from the queue of Deletion QOP, in order to start the execution, therefore it is not a part of deletion QOP queue. Therefore, at Step 93, Reconstruction QOP puts QE by which

checkpoint processing was carried out into the queue of Reconstruction QOP, and a signal is sent to Reconstruction QOP, consequently the execution comes to be continued later by another QOP on another queue to which the QE is moved for the QE in Reconstruction QOP or the back. then, the reconstruction QOP -- CQ, the "QOP Busy PTR" field of the inner deletion QOP is cleared, QE Busy ID in the QE is reset, all relation with Deletion QOP is removed, and it progresses to Step 97 At Step 97, when it goes into Step 90 from an idle condition, it returns to an idle condition. Since it went into Step 91 from step 78N of drawing 9 when that was not right, it returns from Step 97 to step 78B of drawing 9 .

[0098] It is judged by the effective bits contained in the state of "Checkpt Subunit Data" within QE by which checkpoint processing was carried out whether reconstruction occurred during execution of QE. When V bits showed that the effective checkpoint processing information in the QE exists, it occurred. in this case, which operation is possible -- QOP can also complete the execution of QE by using this checkpoint processing information later, and continuing until the QE execution is successfully completed from the subunit of the successful last shown in checkpoint processed data Deletion QOP can cut the power supply. It is because the operation is unnecessary within a subsystem any longer. Then, suppose a part [a different QOP subsystem relevant to other portions of the system which takes out from there while these is exchanged within a QOP subsystem or it is in a subsystem, performs maintenance, or consists of original systems]. QE assigned to the related queue of Deletion QOP is re-assigned to the queue relevant to these QOP(s) in order to route future work to QOP which remains in the subsystem.

[0099] Finally, a reconstruction processor can continue the normal operation by returning to a front state by progressing to Step 97 from Step 93. The reconstruction processing about Deletion QOP is completed at this time.

[0100] As long as QOP in which one operation is possible remains in a system, any number of QOP(s) can be reconfigurated off-line with a continuous off-line change command.

[0101] By the QOP online change command which adds QOP to a subsystem, it progresses to an "additional" path from Step 94. Before QOP added performs processing of drawing 11 , it is physically added to a subsystem.

[0102] At Step 94, the new QOP header of the new queue relevant to new QOP added is set up. At Step 94, QOP added is put into operation by switching on a power supply, if there is need after that, and enabling operation of the QOP queue. New QOP is put on an idle condition. QOP of this state waits for the signal from CPU which shows that QE was put on the queue relevant to it, or the signal from another QOP which moved QE to the queue relevant to it.

[0103] Step 95 -- CQ, when it is alike and QE is contained, it progresses to Step 96 When CQ is empty, it progresses to Step 97. At Step 96, QE of CQ shell head is removed, this QE is put into the queue of added QOP, and a signal is sent to added QOP. At Step 97, when it has progressed from step 78N, it returns to processing of step 78B. Since the reconstruction signal was received at the time of an idle condition, when it progresses to Step 97, it returns to an idle condition and waits for another signal. This subsystem assigns a subchannel automatically and makes new QE put into the new queue added now from now on.

[0104] With a continuous online change command, any number of QOP(s) can be reconfigurated on-line.

[0105] Another method of changing QOP off-line is removing QOP which simulated the permanent fault condition of QOP, used after that the recovery explained in the following paragraph of this specification, and simulated the obstacle.

[0106] Recovery (drawing 12)

It depends for the recovery of drawing 12 on having internally error detection capacity equivalent to each newest processor for which QPO is used commercially now, respectively. If the error situation in QOP is detected, QOP which has an error situation will send a recovery signal to other QOP(s) through QE signal / recovery / reconstruction bus 110 of drawing 13 . If this recovery signal is sent, recovery will begin from Step 100 of drawing 12 . For example, an error situation is detected, when the temporary error exceeded the predetermined number of times of a threshold, or when a permanent error is detected within QOP.

[0107] The first QOP in the state of receiving the recovery demand signal on QE signal / recovery / reconstruction bus 110 detects this signal, and it answers by the acceptance signal which shows that Acceptance QOP works as "recovery QOP" on QE signal / recovery / reconstruction bus 110. It recognizes as what shows that other QOP(s) can continue usual processing operation with them unrelated to recovery action for this acceptance signal sent by Recovery QOP.

[0108] At Step 101, Recovery QOP can reset QOP (Error QOP is called) which generated the obstacle, the increment of the retry count field in the QOP queue header of Error QOP can be carried out, and the number of times of the error generated to Error QOP can be shown.

[0109] Then, Recovery QOP progresses to Step 102 and [whether reset of Error QOP was failure, and] Or retry count (inside of a QOP queue header) detects whether predetermined maximum (N) was exceeded (a subsystem). It can judge whether it is necessary to use the arbitrary methods of including the well-known method by this technical field, and to repeat the processing which the error situation generated about whether Error QOP has a temporary-error state or a permanent error situation and N retry. When the error situation remains also after the Nth retry of the processing, or when reset goes wrong, it is considered that a temporary error is a permanent error situation. N is an experience value set up by experience of processor operation (for example, 20), and is reset periodically. This retry operation is started by the microcode in Recovery QOP which restarts QE processing of Error QOP as shown in Step 103 or Step 104.

[0110] After Step 102, at Step 103, Recovery QOP accesses the header of a common queue and reads the "QOP Busy PTR" field of Error QOP. When an error situation occurs, the address of QE which Error QOP was processing is stored in this field. Step 103 shows that set the "busy ID" field of QE to the identifier of Error QOP, and the QE is using it in the present error QOP after that. Then, Recovery QOP puts this QE into an error QOP queue by setting the "Top Q PTR" field of an error QOP queue header to this address of QE. The chain of the QE is carried out to the head of an error QOP queue by this.

[0111] Then, Error QOP will receive a signal, therefore will progress to step 61A of drawing 7 , and will usually continue processing. At Step 104, Recovery QOP returns to normal operation, and returns to step 78B of drawing 9 , or returns to an idle

condition, and waits for another signal.
[0112] When judged with a permanent error situation at Step 102, it progresses to Step 105 and Recovery QOP performs the remaining steps 105, 106, and 104 of recovery.
[0113] In this case, it progresses to Step 105 from Step 102, and Recovery QOP accesses the header of an error QOP queue, the enable field is set, and a impossible state of operation is shown. Then, Recovery QOP moves all QE(s) in the queue of Error QOP to the queue of Recovery QOP, stops operation of Error QOP, and enables it to perform the maintenance to it. operation in a subsystem is possible for QE assigned to the related queue of Error QOP -- the queue relevant to QOP is re-assigned Then, when Recovery QOP accesses the "QOP Busy PTR" field of the error QOP in a common queue header and generates an obstacle at Step 106, the address of QE which Error QOP was performing is acquired. It is put into this QE at the head of a recovery QOP queue, and Recovery QOP receives a signal, after this "busy ID" field of QE is reset. It is because this QE is not performed now.
[0114] Finally, it progresses to Step 104, and Recovery QOP returns to an idle condition, when a recovery signal is received and it is an idle condition. When that is not right, this recovery is called from step 78M of drawing 9 , therefore returns to step 78B of drawing 9 , and continues processing of Activity QE.
[0115] The checkpoint information in QE which stood by at Step 106 is used by recovery. This QE is performed later on by another QOP in order, when it is moved to QOP queue with this another QE by Recovery QOP again. If the contents of this "Checkpt Subunit Data" field of QE are used, the execution can be restarted from the following subunit after subunit operation finally completed successfully, and it is continuable until execution of the work unit of QE is completed. Although deformation and correction of a large number which do not deviate from the range and the meaning of this invention were shown, probably, it will be clear to this contractor now. Therefore, please understand that an above-mentioned embodiment is offered not as a limit but as an example.
[0116] In addition, as for this invention, it has the work type directions for each work-request item showing an ADM (asynchronous-data movement) work request or an I/O device work request. Each work-request item specifies the parameter for work units of which execution was required. An ADM work unit moves the data page of one or more [unit / work / each] between the positions in electronic memory, or between electronic memory. One or more ADM work subunits controlled by the queue processor are included. It is characterized by being transmitted to an input/output processor by the queue processor, in order that an I/O work unit may control the demanded I/O work. The step which receives a work-request item as a queue element (QE) within two or more processor queues which relate to two or more queue processors in the subsystem of data processing system, respectively, The step which judges whether it tested by the queue processor during execution of a work unit, and the new work-request item was received on the related processor queue as a receipt queue, To the input/output processor specified when the specified input/output processor was usable to execution of a new I/O work-request item Moreover, the step which transmits the new I/O work-request item found at the test step to a standby input/output queue from a receipt queue when use of the specified input/output processor is impossible, The queue processor which tests is using it in an ADM work unit. The step which moves the new ADM work-request item found at the test step to other processor queues from a receipt queue while other queue processors in a subsystem are not using [be / it] it, When the ADM work-request item which it is not, and a queue processor is not using it, and is not served on a related queue exists, for another ADM work-request item which exists on a related processor queue The load-balancing method containing the step which makes each queue processor start execution of another ADM work unit which controls the asynchronous workload of data move operation between the queue processors of the plurality in the subsystem of data processing system.
[0117] A test step swerves from execution of an ADM work unit temporarily further. Inspect the header of a related processor queue about directions of the new ADM item received on the queue, or a newcomer output item, and the common control block which includes a state item during use of all the queue processors in a subsystem is inspected. The load-balancing method given in 0116 characterized by including judging whether other queue processors can process not under the present use but a new ADM item.
[0118] The load-balancing method given in 0117 characterized by including the step which moves a new work business item to an accessible common queue from all the processors in a subsystem when the transfer step is further shown that all the queue processors in a subsystem are states during the present use.
[0119] The load-balancing method given in 0118 characterized by including the step to which a transfer step starts further execution by one of the queue processors of the ADM work-request item which exists on a common queue between execution of an ADM work unit.
[0120] Furthermore, a common queue header is constituted so that it may have the pointer field during queue processor use of the plurality for storing the address to the ADM work-request item performed by each queue processor in a subsystem now, respectively. The step which relates a common queue header with a common queue, and the step which sets the pointer field to the address of the ADM work-request item under execution by the processor during the processor use for related queue processors, While the ADM work request is performed by the related queue processor The load-balancing method given in 0118 to which both the set step is characterized by performing at the time of the start of each ADM work subunit including the step which sets the state field during each processor use in a common queue header.
[0121] The preparation for processing according [a subsystem] to the processor in a subsystem which is not expected and which interrupts and carries out shell recovery should do. Furthermore, in order to enable it to continue execution of an ADM work-request item by which another queue processor was partially performed using the contents of a checkpoint data field, The step which constitutes each ADM work-request item so that it may have a checkpoint data field for storing the checkpoint information about the ADM work subunit finally successfully completed by the queue processor is included. The load-balancing

method given in 0120.
[0122] The load-balancing method of the publication by 0121 in which the preparation for processing according [a subsystem] to the processor in a subsystem which is not expected, and which interrupts and carries out shell recovery is made, and the content of a checkpoint data field contains the step which shows another queue processor whether it is usable, and which prepares effective bits in a checkpoint data field in order to continue at the last execution of an ADM work-request item by which the composition step was performed still more nearly partially from the ADM work subunit by which checkpoint processing was carried out.
[0123] The preparation whose subsystem recovers the obstacle by the processor in a subsystem is made. a composition step further An effective-bits subfield, The control word subfield which stores the address to the head of an ADM program specified according to the ADM work-request item, The designated-block address subfield which stores the address of designated block which specifies the subunit control parameter used by the related ADM program, The number subfield of subunits which stores the number of the subunits successfully moved before the ADM work-request item under present execution is included. The load-balancing method given in 0121 characterized by including the step which prepares two or more subfields for storing the processed [checkpoint] subunit data in each ADM work request in each checkpoint data field.
[0124] The preparation whose subsystem recovers the obstacle by the processor in a subsystem should do. Furthermore, the step which sends an error indication signal when an error situation occurs during the processing which performs the present ADM work-request item on one of queue processors, The step which retries a part of processing [at least] which the error situation generated, After completion of a retry step performs recovery of the present ADM work-request item on other queue processors, when an error situation continues existing. on other queue processors The step which processes the present ADM work-request item from the subunit successfully completed at the last shown by the checkpoint data field of the present ADM work-request item, The load-balancing method given in 0123 containing the step which continues processing of the present work-request item on the queue processor which has an error situation when an error situation stops continuing existing in a retry step.
[0125] Furthermore, a common queue header is accessed by the queue processor at the time of completion of processing of the ADM unit of the work by the queue processor. The step which performs the item when it judges whether an ADM work-request item exists on a common queue and an item is found on a common queue, When an ADM work request does not exist on a common queue, at the time of completion of processing of the ADM unit of the work by the queue processor The step which accesses the following ADM work-request item from a related queue, and performs this by the queue processor, The load-balancing method given in 0123 which contains the step which makes a queue processor an idle condition when a work item is not found on the queue relevant to a common queue top or a queue processor.
[0126] Furthermore, the load-balancing method given in 0125 which contains the step which performs the state field, the step which clears the state field during the use within an ADM work-request item, and the step which send the signal which shows completion of ****** of an ADM work unit to arithmetic and program control during use of the related processor in a common queue header at the time of the step which performs each work unit of the present ADM work-request item removed from the related queue by the queue processor, and completion of each work unit.
[0127] Furthermore, the load-balancing method given in 0124 containing the step which inspects a reconstruction signal after completion of each ADM work subunit, and the step which reconfigurates a subsystem when a reconstruction signal is received by the subsystem.
[0128] Furthermore, the load-balancing method given in 0123 containing the step which accepts a reconstruction signal by the queue processor of a state during un-using [which is specified by the reconstruction processor by sending an acceptance signal to other queue processors in a subsystem] it, and the step which detects the specification which shows which queue processor is added or deleted by the subsystem from a reconstruction signal by the reconstruction processor.
[0129] The load-balancing method given in 0128 characterized by an acceptance step containing further the step which performs an acceptance step between execution of the subunit of the work by the reconstruction processor.
[0130] Furthermore, the load-balancing method given in 0128 containing the step which moves all work-request items to the queue relevant to the queue processor in which operation in a subsystem is possible by the reconstruction processor from the step which makes the queue relevant to the queue processor specified that it should remove from a subsystem impotentia of operation by the reconstruction processor, and the related queue of the queue processor removed.
[0131] Furthermore, the load-balancing method given in 0130 containing the step which traces the ADM work-request item relevant to the queue processor specified that it should remove when the execution of the work unit of the ADM work-request item relevant to the queue processor specified that it should remove by the queue processor specified that it should remove was not completed in that the processing was suspended by other processors by other queue processors.
[0132] Furthermore, the load-balancing method given in 0131 which uses the checkpoint data field relevant to the traced ADM work-request item, and contains the step which continues execution of the item by another queue processor from the subunit finally performed successfully.
[0133] Furthermore, the load-balancing method given in 0127 containing the step which writes the step which constitutes the queue header of the queue relevant to the queue processor added to a subsystem, and the identifier of the queue processor added to a subsystem since a queue processor is related with the queue expressed by the queue header in a queue header.
[0134] The step which receives the work-request item which specifies the parameter about the work unit of which execution was required by the queue processor in a subsystem within two or more queues which relate to two or more queue processors, respectively, So that a work unit may be performed by performing one or more work subunits The step which performs the work-request item on the queue relevant to a processor by each queue processor, The step which performs the test about the new

work business demand item received on the queue relevant to a processor during execution of the present work-request item during the pause of execution of the present work-request item by the processor, When it is shown that other queue processors are not running states, the new work business item which it made clear by the test step that it is on the queue relevant to a queue processor is moved to other queues. When all other queue processors in a subsystem are in a running state, To an accessible common queue, each of that item from all the queue processors in a subsystem The load-balancing method containing the step which moves a new work business item which controls the asynchronous workload of data move operation between the queue processors of the plurality in the subsystem of data processing system.

[0135] Furthermore, the load-balancing method given in 0134 containing the step which executes the instruction for putting a work-request item on the queue of a subsystem with the arithmetic and program control (CPU) in data processing system as a queue element which stores the information which specifies the work unit performed by the subsystem, and the step which sends the signal which shows that the queue element was put on the queue by CPU to the queue processor relevant to a queue.

[0136] Furthermore, in order to reconfigurate a subsystem, it carries out a chain to the queue in a subsystem, using a work-request item as a queue element.